`flankr`: An R package implementing computational models of attentional selectivity

James A. Grange
School of Psychology, Keele University, UK

Abstract

The Eriksen flanker task (B. A. Eriksen & Eriksen, 1974) is a classic test in cognitive psychology of visual selective attention. Two recent computational models have formalised the dynamics of the apparent increasing attentional selectivity during stimulus processing, but with very different theoretical underpinnings: The shrinking spotlight (SSP) model (White, Ratcliff, & Starns, 2011) assumes attentional selectivity improves in a gradual, continuous manner; the dual stage two phase (DSTP) model (Hübner, Steinhauser, & Lehle, 2010) assumes attentional selectivity changes from a low- to a high-mode of selectivity at a discrete time-point. This paper presents an R package—`flankr`—that instantiates both computational models. `flankr` allows the user to simulate data from both models, and to fit each model to human data. `flankr` provides statistics of the goodness-of-fit to human data, allowing users to engage in competitive model comparison of the DSTP and the SSP models on their own data. It is hoped that the utility of `flankr` lies in allowing more researchers to engage in the important issue of the dynamics of attentional selectivity.

Keywords: Flanker task; attentional selectivity; computational model; R

The Eriksen flanker task (B. A. Eriksen & Eriksen, 1974) is a classic test of visual selective attention in cognitive psychology. In this task, subjects are (typically) presented with a string of arrows (e.g., <<><<) and it is the subjects' task to judge whether the central arrow is facing left or right, and make a response accordingly. The experimenter manipulates whether the flanking stimuli are incongruent or congruent with respect to the target's direction: incongruent flankers face in the opposite direction to that of the target (e.g., <<><<), and congruent flankers face the same direction as the target (e.g., >>>>>). It is a consistent finding that response times (RTs) and error rates are increased on incongruent trials compared to congruent trials. This congruency effect reflects the

Please address correspondence to James A. Grange, School of Psychology, Dorothy Hodgkin Building, Keele University, Keele, UK, ST5 5BG. Email: grange.jim@gmail.com. I am extremely grateful to Ronald Hübner for many discussions regarding the fitting procedures used in this package, and two anonymous reviewers for excellent recommendations to improve the package and manuscript.

interference of distracting information on visual selective attention.

Despite this interference, subjects are still able to perform with good accuracy on incongruent trials, suggesting the negative effects of the flankers can be overcome with sufficiently focussed selective attention; that is, selective attention mechanisms must operate in such a manner to enable subjects to focus on the central target rather than on a distracting flanker. Thus, the flanker task is an ideal tool with which to explore the characteristics of attentional selectivity.

Processing in the Flanker Task

Performance in the flanker task has been explained with recourse to a "spotlight" metaphor of attention (Heitz & Engle, 2007; Jonides, 1983) which is able to "zoom" in on the central target to reduce the effect of the flankers. This narrowing of the attentional focus takes time, and as such it is typically found that attentional selectivity improves with time (C. W. Eriksen & St.James, 1986). Gratton, Coles, Sirevaag, and Eriksen (1998) utilised the flanker task together with response time distribution data to explore the dynamics of attentional selectivity. Gratton et al. (1998) used so-called conditional accuracy functions (CAFs), which divides subjects' data into bins or quantiles—from fastest responses to slowest responses—and plots the accuracy for each bin (see the right panel of Figure 6 of the current paper for an example of a CAF). Gratton and colleagues observed that in the fastest bins, a large congruency effect in the error rates was present, but that this error congruency effect reduced at the slower RT bins; this reduction in the error congruency effect was primarily driven by a reduction of errors for incongruent trials at slower responding speeds. This finding is consistent with the idea that attentional selectivity increases with time: at stimulus onset, selectivity is relatively poor, such that response selection is influenced by the central target and the flankers; as processing time increases, attentional selectivity improves such that the influence of the flankers on response selection is reduced, and response selection is primarily driven by the central target.

There has been a surge of interest recently as to the nature of this increase in attentional selectivity. Some authors have suggested that attentional selectivity increases in a continuous and gradual fashion (Heitz & Engle, 2007; White et al., 2011), much like the gradual decrease of the focus/diameter of a spotlight as it becomes intensified (i.e., the zoom-lens metaphor). Other authors have suggested that attentional selectivity improves in a discrete, stage-like, manner (Hübner et al., 2010; Hübner & Töbel, 2012), with attentional selectivity being relatively poor in a first stage of processing, but switching to a more focussed processing mode at a discrete point in time.

The dynamics of the increase in attentional selectivity is an important question if we wish to understand how selective visual attention operates. As these two theories discussed are hard to disambiguate at the behavioural level, this question has begun to be tackled using formal computational models. Computational models are advantageous for theory development and testing for a number of reasons (Farrell & Lewandowsky, 2010; Fum, Missier, & Stocco, 2007; Grange & Houghton, 2014; Lewandowsky & Farrell, 2010), but come into their own when arbitrating between competing accounts of the same data, as the precise, quantitative (cf., verbal models), model predictions can be directly compared to observed human data. Competitive model comparison techniques can then be used to select the best-fitting model, whilst accounting for model complexity.

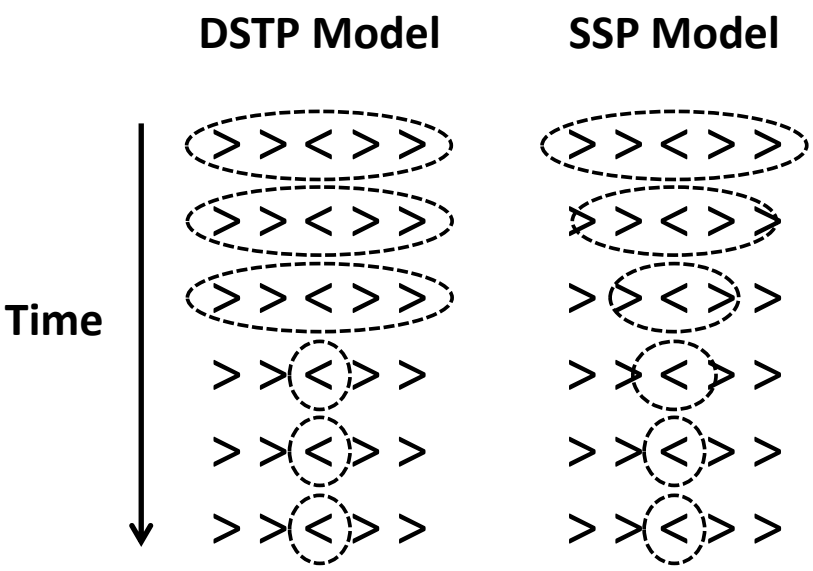


Figure 1. Focus of attentional selectivity (depicted by the dashed ellipse) in a flanker task in the dual stage two phase (DSTP) model and the shrinking spotlight (SSP) model.

Two recent successful formal implementations of the dynamics of attentional selectivity in the flanker task are the dual stage two phase (DSTP) model of Hübner et al. (2010), and the shrinking spotlight (SSP) model of White et al. (2011). Both models assume that attentional selectivity improves with time: at early stages of processing, the response selection process is influenced by the central target and the flankers, whereas later in processing response selection is influenced solely by the central target. However, both models hold different assumptions on the dynamics of this shift in processing.

The opposing assumptions are depicted schematically in Figure 1. Time progresses from top-to-bottom, and the components of the stimulus display influencing response selection processes at each time point is captured by the dashed ellipse. In the DSTP model, response selection occurs in two distinct phases: in a first phase, attentional selectivity is rather poor, so response selection is influenced by the target and flankers; however, at a discrete point in time, late attentional processes manage to select a single stimulus—either the central target or a flanker—and response selection enters a second stage which is only influenced by the selected stimulus. Response selection in the SSP model is also influenced by attentional selectivity. In contrast to the DSTP model, attentional selectivity is thought to improve in a continuous—rather than discrete—fashion: as attention "zooms in" on a central target, response selection becomes more influenced by the central target and less by the flankers.

In their publication of the DSTP model, Hübner et al. (2010) compared fits of their model with several variants of continuous-improvement models and found the DSTP ac-

counted for the data better than any continuous model. White et al. (2011), however, proposed the SSP model and compared the SSP model with the DSTP across several experiments, and found the SSP model fit flanker data better than the DSTP model. Hübner and Töbel (2012) addressed this apparent discrepancy between the studies' findings, and found that the difference in model support was primarily governed by differences in the experimental stimuli and design between studies; in particular, Hübner and Töbel (2012) concluded that the response–stimulus interval—the time between a response to one stimulus and the onset of the next stimulus—was responsible for differences in model superiority. They concluded that more studies are required to conclusively arbitrate between the two accounts.

Overview of the Paper

The purpose of the present paper is to provide researchers with software with which to utilise the DSTP and SSP models to engage with this debate as to the dynamics of attentional selectivity. In the next sections, the formal computational details of each model are provided. I then introduce an R statistics (R Core Team, 2014) package—`flankr`—that implements both of these models. I describe how to install `flankr` and R, and then discuss the core functionality of `flankr`, including how to simulate data from the SSP and DSTP model, and how to fit each model to human data. It is hoped that `flankr` will enable researchers to explore these accounts of attentional selectivity in the flanker task, and to be able to apply them to their own data.

Dual Stage Two Phase (DSTP) Model

In the DSTP model, response selection proceeds as a diffusion process with two absorbing response boundaries (see upper panel of Figure 2): if the diffusion process for response selection reaches the upper boundary (the height of which is set by the parameter A), this signifies a correct response from the model; the diffusion process reaching the lower boundary (the height of which is set by the parameter $B = -A$) signifies an error from the model.

In early stages of processing, attentional selectivity is poor, and therefore response selection is influenced by both the flankers and the central target. Formally, the drift rate—that is, the rate of evidence accumulation towards a particular response—in this early phase is the sum of model parameters, μ_{TA} and μ_{FL}, reflecting the contribution of the central target and flankers, respectively, to response selection. If the current stimulus is incongruent, μ_{FL} is set negative.

In parallel to the first stage of response selection, late attentional processes work to select a single item from the stimulus array for further processing. The time it takes for this stimulus selection to occur is also modelled by a diffusion process (see lower panel of Figure 2), again with two absorbing boundaries. The drift rate for this stimulus selection process is parametised by μ_{SS}. If this stimulus selection process reaches the upper boundary (the height of which is set by the parameter C), it is assumed the model has selected the central target; if the stimulus selection process reaches the lower boundary (the height of which is set to $D = -C$), it is assumed the model has erroneously selected one of the flankers for further processing.

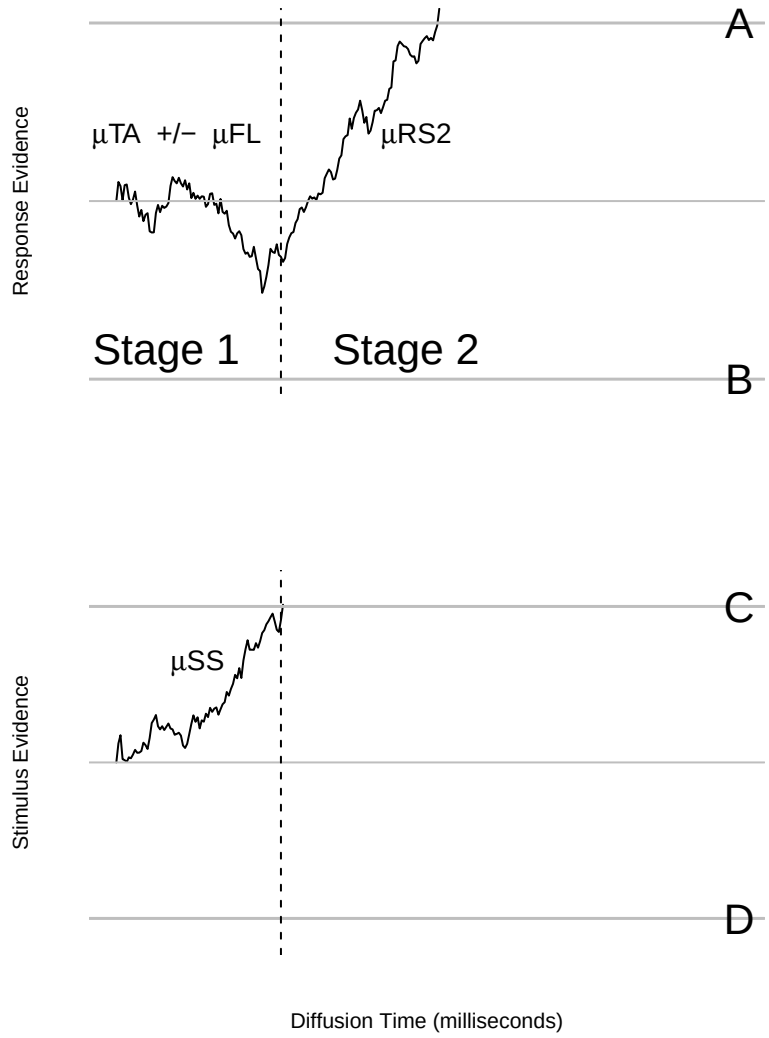


Figure 2. Schematic of response-selection processes in the dual stage two phase model model. The upper panel shows the time-course of a drift rate for response selection towards the correct response (upper boundary); the lower panel shows the time-course of attentional selection mechanisms for selecting to process the target (upper boundary) instead of erroneously selecting to process a flanker (lower panel). When stimulus selection has finished, response selection enters its second stage.

If this stimulus selection process finishes before response selection has finished, response selection enters its second—highly selective—stage where the drift rate is determined by which stimulus was selected: if the central target was selected (as is the case in Figure 2), drift rate for the second stage of response selection is set with the model parameter μ_{RS2}; if, however, the stimulus selection process erroneously selects a flanker, and the stimulus is incongruent, the drift rate for the second stage of response selection is set to $-\mu_{RS2}$. The point in time stimulus selection finishes and response selection enters its second stage is depicted by the vertical dashed line on Figure 2.

The final parameter in the DSTP model is *ter*, which reflects the time taken for stimulus encoding and motor responding time. All parameters for the DSTP model are shown in the upper section of Table 1.

Table 1
List and brief description of model parameters in the dual stage two phase (DSTP) model and the shrinking spotlight (SSP) model.

Model	Parameter	Description
DSTP	A/B	Height of response selection boundary
	C/D	Height of stimulus selection boundary
	μ_{TA}	Drift rate for central target during response selection stage 1
	μ_{FL}	Drift rate for flankers during response selection stage 1
	μ_{SS}	Drift rate for stimulus selection
	μ_{RS2}	Drift rate for stage 2 of response selection
	ter	Non-decision time
SSP	A/B	Height of response selection boundary
	ter	Non-decision time
	p	Perceptual input of stimuli
	rd	Rate of attentional distribution reduction
	sd_a	Initial width of attentional distribution

Shrinking Spotlight (SSP) Model

Response selection in the SSP model is also modelled as a diffusion process with two absorbing response boundaries (parametised by A, and $B = -A$). Each element of the stimulus display provides perceptual evidence, p, for a particular response; this value can either be positive or negative, depending on the direction of the arrows in the stimulus display. For generality, we can assume that all elements in the display have perceptual input p if the stimulus is congruent; if the target is incongruent, the central target takes a positive value for p and the flankers take a negative value for p.

The drift rate, $v(t)$, at each point in time, t, in the response selection process is a combination of the strength of perceptual input for each element in the stimulus array multiplied by the proportion of total attention currently being paid to each element. The distribution of attention over the stimulus display changes with time, with more attention being paid to a central single item as time progresses (i.e., selectivity on the central target

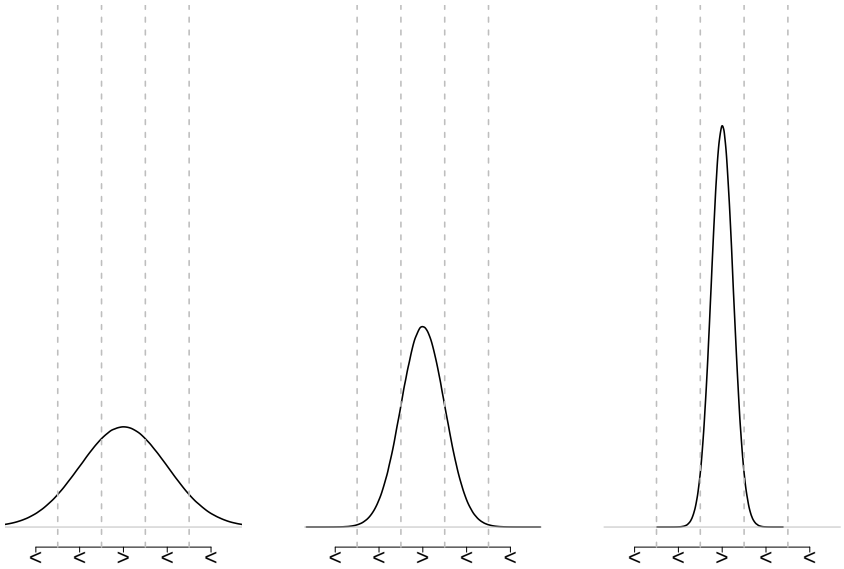


Figure 3. Schematic of the shrinking spotlight model. Time moves from left to right, and the normal distribution represents the proportion of attention paid to each element in the stimulus array as time progresses. In this example, attentional focus becomes stronger on the central target arrow. Adapted from White & Ratcliff (2011).

improves with time). Consider the far left panel of Figure 3, which shows how attention is initially distributed across the stimulus display at stimulus onset. White et al. (2011) modelled the distribution of attention over the stimulus display as a normal distribution centered on the target. (In this example, the central target is positioned over zero, and each element of the stimulus display is considered to be one unit wide.) The initial width of the attentional spotlight—i.e., the standard deviation of the normal distribution—is given by sd_a.

As time progresses, the width of the attentional spotlight decreases; formally, the standard deviation of the normal distribution over the stimulus display decreases such that the standard deviation at the current time, $sd_a(t)$ is given by

$$sd_a(t) = sd_a - r_d t, \tag{1}$$

where r_d is the rate at which the spotlight is shrinking at time t. The reduction in $sd_a(t)$ is clipped to a minimum of 0.001. From these assumptions, the total attention, a, being paid to the outer flankers (a_{outer}), the inner flankers (a_{inner}), and the central target (a_{target}) at the current time, t, is calculated by

$$
\begin{aligned}
a_{outer}(t) &= \int_{1.5}^{\infty} \phi[0, sd_a(t)]; \\
a_{inner}(t) &= \int_{0.5}^{1.5} \phi[0, sd_a(t)]; \\
a_{target}(t) &= \int_{-0.5}^{0.5} \phi[0, sd_a(t)].
\end{aligned} \tag{2}
$$

In Equation 2, ϕ is the density function for the normal distribution with mean 0 and standard deviation of $sd_a(t)$ (Equation 1). As time progresses, the width (indexed by the standard deviation) of the spotlight reduces, which reduces the impact of the flankers on drift rate.

As total drift rate for the response selection process is a combination of the perceptual input, p, of each element in the display multiplied by the attention, a_i, currently being paid to each element, the drift rate at each time point, $v(t)$, is given by

$$
v(t) = 2p_{outer}a_{outer}(t) + 2p_{inner}a_{inner}(t) + p_{target}a_{target}(t). \tag{3}
$$

As with the DSTP model, the SSP model has a non-decision parameter, ter, which captures stimulus encoding and motor responding time. All parameters of the SSP model can be seen in the lower portion of 1.

Using `flankr`

This section provides an overview of how to use `flankr`, from installing the software, to fitting some data, and through to plotting the outcome of the model fitting routine.

Installing `R` and `R-Studio`

The package `flankr` requires a working version of R statistics (R Core Team, 2014)[1]. R is a free software environment and programming language designed for statistical computing, and is available for Windows, Mac, and Linux users. To make editing scripts in R more manageable, the author recommends the user download R-Studio[2], a free user interface (i.e., IDE) for R. Once R and R-Studio are installed, open R-Studio. R-Studio is divided into four panes (see Figure 4; if pane 1 is not visible to the user, go to `File--New File--R Script`).

Pane 2 is the command line where R is provided commands to perform by the user. All commands provided as examples in this paper can be inputted directly to this pane. Pane 1 is useful if the user has multiple lines of commands; this pane allows users to write scripts should they so wish (see examples in the Discussion), although for basic functionality of `flankr` this is not necessary.

[1] R can be downloaded for free from http://www.r-project.org/.

[2] Available from http://www.rstudio.com/

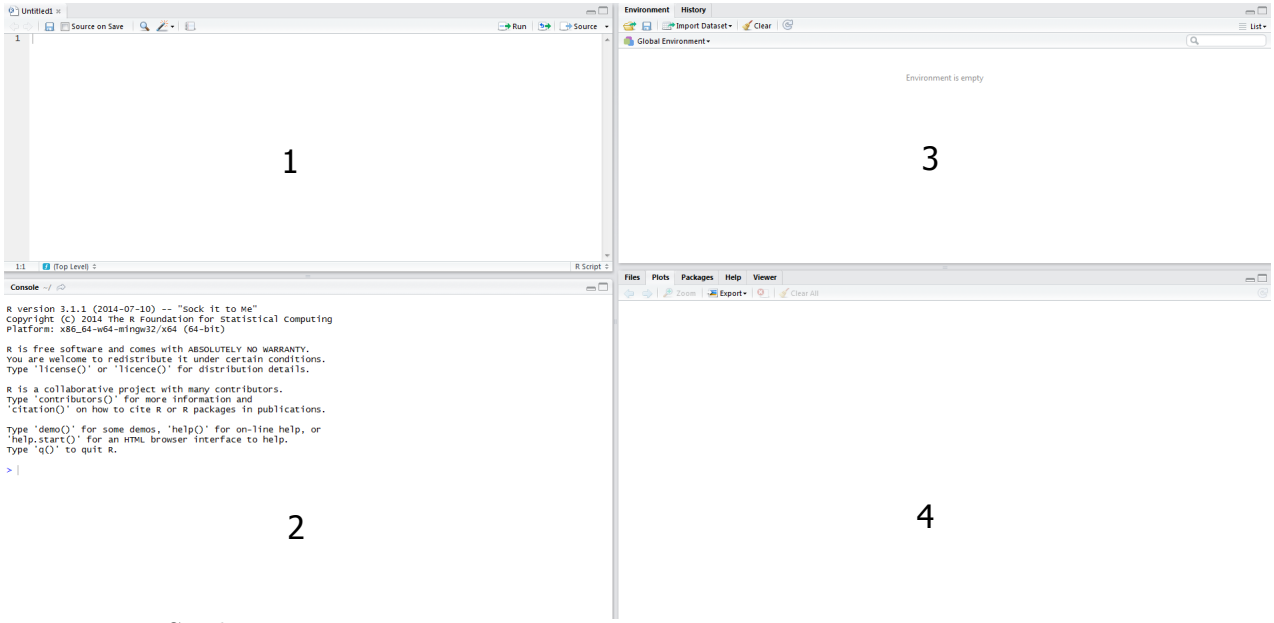


Figure 4. R-Studio overview

Installing `flankr`

The "base" R installation can be supplemented by installing so-called "packages", which provide unique functions designed to address some particular statistical/programming need. `flankr` is one such package, although it is hosted by the author in a GitHub repository[3]. `flankr` is written in R and C++ (using the RCPP package; Eddelbuettel & François, 2011). Before installing `flankr`, the user must install `devtools`, a development package which allows installation of packages hosted on GitHub (as `flankr` currently is). In the command line, type

```
> install.packages("devtools")
```

This will install `devtools` and all of its dependencies onto the user's system. Once this has completed, `flankr` can be installed by typing the following into the command line:

```
> devtools::install_github("JimGrange/flankr")
```

Once installed, now type the following to enable the functionality of `flankr` for the current session (note that this will need to be entered into the command line every time the user wishes to use `flankr` in a new session):

```
> library(flankr)
```

[3] Although the user will not need to visit this repository, it is located at https://github.com/JimGrange/flankr for the interested reader. The repository includes copy of all of the code for both visible functions (i.e., ones accessible to the user) as well as invisible functions, which run behind the scenes.

Simulating Synthetic Data

The user is able to simulate synthetic data for both the DSTP and the SSP model by using the `simulateDSTP` and `simulateSSP` functions, respectively. These functions produce synthetic data in the form of response time and accuracy for *both* congruent and incongruent trials; the user need-not (and in fact cannot) simulate data separately for each congruency level. Also note that—to my knowledge—neither the SSP nor the DSTP models have been fit to neutral trials, and it is an open theoretical question how they would explain processing during neutral trials. Thus, at present, `flankr` only allows simulation/fitting of congruent and incongruent trials.

The functions take only three arguments: `parms` declares what model parameters the user would like to generate (in the order as presented in Table 1); `nTrials` sets how many trials the user would like to simulate for each congruency level; `seed` is an optional argument which sets R's random number generator to a particular state which allows reproducible simulations (this is initiated to NULL so that new simulated data is returned each time). To simulate the DSTP model, and store the data into a variable called `dstpData`, enter the following:

```
> parameters <- c(0.070, 0.086, 0.045, 0.065, 0.368, 1.575, 0.225)
> dstpData <- simulateDSTP(parms = parameters, nTrials = 1000)
```

Simulating the SSP model follows a similar procedure, except the call is to a different function, and uses parameters characteristic of the SSP model:

```
> parameters <- c(0.050, 0.300, 0.400, 0.040, 1.500)
> sspData <- simulateSSP(parms = parameters, nTrials = 1000)
```

To see the data just simulated, type into the console the name of the variable the data was stored into. To see just a portion of the data, type `head(variableName)`, where "variableName" pertains to the name of the variable used to store the data. Whichever model is being simulated, the function returns a data frame with three columns: response time (rt) in seconds, accuracy, and the congruency of each trial.

Fitting Human Data

This section describes how to use `flankr` to fit the DSTP and SSP models to human data. Primarily, the DSTP model and the SSP model have been fit to data that just differ on the congruency of experimental stimuli (i.e., incongruent and congruent trials). To increase the generality of `flankr`, the package can accept data structures that contain additional factors (for example, the "presence" or "absence" of a warning tone before stimulus onset).

Note that if additional conditions (i.e., other than congruency) are to be fit, the models can only be fit to one additional condition at a time; that is, individual conditions (e.g., "present" and "absent") must be passed to `flankr` one at a time, and thus separate parameters will be estimated for each additional condition. For each additional condition (or if the user does not have an additional condition), the model fits congruent and incongruent trials *simultaneously*; that is, the user will receive one set of best-fitting parameters—that jointly describe congruent and incongruent trial performance—for each additional condition examined.

	A	B	C	D	E
1	subject	condition	congruency	accuracy	rt
2	1	absent	incongruent	1	0.591
3	1	absent	incongruent	1	0.533
4	1	present	incongruent	1	0.485
5	1	present	incongruent	1	0.583
6	1	present	incongruent	1	0.656
7	1	absent	congruent	1	0.607
8	1	present	congruent	1	0.684
9	1	present	incongruent	1	0.678
10	1	present	congruent	1	0.498
11	1	present	incongruent	1	0.66
12	1	absent	congruent	1	0.529
13	1	absent	congruent	1	0.595
14	1	absent	congruent	1	0.586
15	1	present	congruent	1	0.538
16	1	present	incongruent	1	0.654
17	1	absent	congruent	1	0.631
18	1	absent	incongruent	1	0.849
19	1	absent	congruent	1	0.637
20	1	absent	congruent	1	0.714
21	1	present	incongruent	1	0.758

Figure 5. Example of how data should be organised in the .csv file.

Figure 5 shows how data must be organised within a .csv file[4]. Note that all characters in the data must be in lowercase exclusively. (Note that R is a case-sensitive language.) The `subject` column codes for the subject number of all subjects in the experiment. The examples presented in the main of this paper will discuss model fits to data from multiple subjects; for details how to fit the models to individual subjects, please refer to the help files for the fitting functions (see e.g., Tables 2 and 4). `condition` is an optional column with text coding for the levels of an additional factor in the experiment. All additional conditions in the experiment must be coded in this column, so if the user has more than one additional factor, they must all be coded in this column (e.g., "a1", "a2", "b1", "b2"). In this example data, the experiment manipulated whether a warning signal was present or absent before the onset of the flanker stimulus. The `congruency` column codes the congruency of the current stimulus, and only accepts two levels (congruent & incongruent). The `accuracy` column codes the accuracy of the current response, with `1` coding a correct response and `0` coding an error response. `rt` codes the response time (in seconds) for the current response.

The data set shown in Figure 5 is included in the `flankr` package. To activate this data, type

```
> data(exampleData)
```

The data is now available in the variable `exampleData`. If the user wishes to use their own data, users familiar with .csv input into R can use standard procedures, and those new to R can use the following to load their data into a variable called `myData`:

[4] At present, `flankr` only accepts .csv files, unless the user is familiar with how to input other data formats into R such that R accepts it as a data frame.

```
> myData <- getData()
```

This will launch a dialog box which will allow the user to orient to where their data is stored on their system. The examples in this paper will use the `exampleData` data provided with the package.

The Fitting Procedure. Before discussing the functions used for fitting each model, this section will introduce the model fitting routine. The fitting procedure involves finding a set of parameters that best fits the human data, in particular to response time distribution information. The response time distribution information for the human data is presented in the form of cumulative distribution functions (CDFs) for correct responses and conditional accuracy functions (CAFs) to account for accuracy performance.

CDFs are constructed by finding the quantile cut-off points for each individual subject separately for both congruency conditions; by default, `flankr` uses the 0.1, 0.3, 0.5, 0.7, and 0.9 quantiles. The quantile cut-off values are then averaged across subjects, again separately for each condition. Example CDFs for human data can be seen as circles in the left panel of Figure 6. As can be seen, CDFs allow presentation of how the congruency effect changes across the whole of the response time distribution.

CAFs are constructed by dividing each subject's entire data—both correct and error trials—into bins; in the current example, four equal sized bins are used, each containing 25% of data, separately for each congruency condition. The number of bins `flankr` uses is set to four by default, but again this can be changed by the user. The mean response time and percent accuracy is then calculated for each of the bins; these values are then averaged across subjects. Example CAFs for human data can be seen as circles in the right panel of Figure 6. CAFs are advantageous for presenting data in an attentional selectivity task (such as the Flanker task) as it portrays how accuracy improves as response time increases, showing the tendency for improved selectivity with time.

The model fitting routine attempts to find parameters that produce synthetic data that matches as closely as possible the proportion of responses in each CDF and CAF bin that was found in the (averaged) human data separately for each congruency level.

For example, with perfect accuracy, a congruent condition with CDFs of 0.1, 0.3, 0.5, 0.7, and 0.9 produces 6 bins for correct response time, with 10%, 20%, 20%, 20%, 20%, and 10% of responses in each bin, respectively, with no responses in the CAF bins (as no errors occurred). However, accuracy is rarely perfect, so the proportion in each bin is scaled by the accuracy. With 92% accuracy, for example, the proportion of data in the second CDF bin—between the 0.1 and 0.3 quantiles—is (0.3 - 0.1) * 0.92 = 0.184. Then, the proportion of error responses found in the CAF bins is calculated. Note that the proportion of correct responses distributed across the CDFs and the proportion of error responses distributed across the CAFs must sum to 1 (separately, for each congruency condition). All of these calculations for the human data and the model predictions are handled behind the scenes by `flankr`.

The degree of fit between the human data and the model prediction is assessed by the likelihood ratio chi-square statistic, G^2, which is given by

$$G^2 = 2\sum_{i}^{J} N p_i ln\left(\frac{p_i}{\pi_i}\right), \tag{4}$$

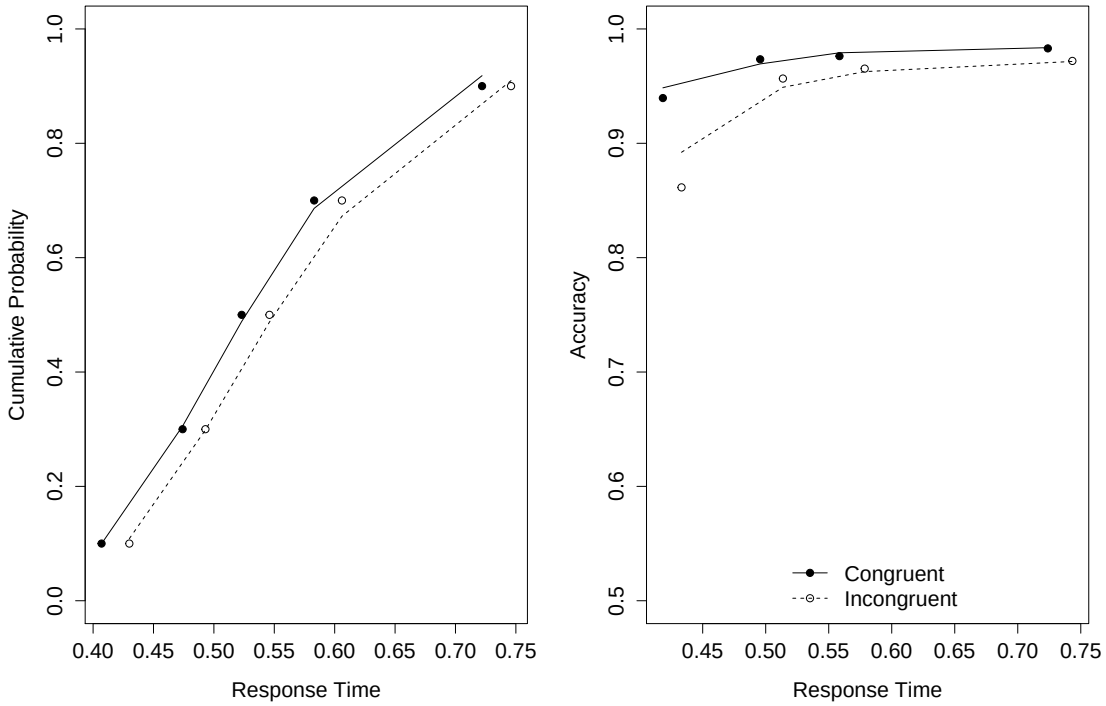


Figure 6. Cumulative distribution functions (left panel) and conditional accuracy functions (right panel) of human data (circles) for congruent and incongruent trials. The lines represent an example of a fit of the DSTP model to the human data.

where p_i is the proportion of human observations in the ith bin, π_i is the proportion in this bin predicted by the model, N is the average number of trials, J is the total number of bins, and ln is the natural logarithm. Note that N in `flankr` is always set to 250, and thus G^2 should not be used for significance testing; it is rather used solely as an objective function by `flankr` to aid finding parameters that best-fit the human data. As such, comparison of G^2 values across models fit to different data sets is not meaningful.

To aid comparison of the fit to human data between the DSTP model and the SSP model, `flankr` also provides the Bayesian Information Criterion (BIC) statistic. The BIC statistic reflects goodness of fit whilst also accounting for the complexity of the model being considered. Model complexity increases with the number of parameters in the model; more complex models are also more flexible, and—with all other things being equal—will fit data better than a model with fewer parameters. The BIC includes a penalty term for the number of parameters in a model; thus, when deciding between competing models, the model with the lowest BIC value is to be preferred.

`flankr` utilises the BIC for binned data—denoted here by bBIC—reported by Ratcliff and Smith (2004) which is calculated by

$$\text{bBIC} = -2\left(\sum_{i}^{J} Np_i ln(\pi_i)\right) + Mln(N), \tag{5}$$

where M is the number of free parameters in the model being considered, and all other terms are the same as in Equation 4.

Parameter Optimisation Routine. `flankr` uses parameter optimisation routines found in R's `optim` function, using a graded decent method; specifically, the Nelder-Mead method is used to find the best fitting parameters that reduces the G^2 statistic until no more improvements are found (i.e., it reaches a minimum). On each iteration of the parameter optimisation routine, `flankr` simulates a set number of trials from the model being fit using the parameters currently being considered by the `optim` function. The proportion of model data found in each CDF and CAF bin is then compared to that found in the human data using the G^2 statistic; `optim` then generates new parameters that aim to reduce this statistic.

In a later section, I discuss potential issues regarding parameter optimisation (i.e., that of avoiding so-called "local minima") that are not unique to the models implemented in `flankr`, and I describe methods for avoiding these issues as best as possible. In the next section, I describe how the DSTP and SSP models can be fit to data using the `flankr` package, and how to visualise the fit of the model. Detailed instructions are provided for how to fit the DSTP model to human data. The functions that enable the fits are practically identical between the DSTP model and SSP model, so only one will be discussed in detail.

Fitting the DSTP Model. The DSTP model comprises 7 parameters (see Table 1) which need to be estimated from the human data. Functions that employ the DSTP model are outlined in Table 2. The fitting procedure is implemented via the `fitDSTP` function, which needs to be passed several arguments, some of which are optional, and some of which are mandatory. All of the arguments are shown in Table 3 with a brief description of each.

Table 2
Overview of the functions associated with the DSTP model. To explore these functions in more detail, type ?functionName into the R console, where "functionName" is the name of the function detail is required for (e.g., ?fitDSTP).

Function Name	Explanation
simulateDSTP	Generate simulated data from the DSTP model. Returns RTs (in seconds) and Errors for congruent and incongruent trials.
fitDSTP	Fit the DSTP model to human data. Returns the best-fitting parameters, as well as the G^2 statistic and the bBIC statistic.
plotFitDSTP	Plot the fit of the DSTP model. This returns a plot of CDF and CAF for the data being fit, as well as that of the model with the best-fitting parameters found via the fitting routine. This function also returns distributional information from the human data and the model fit, so the user can use their own plotting software.
fitMultipleDSTP	Fits the DSTP model using multiple parameter starting points. The function returns the best-fitting parameters found via the search.

Table 3
Arguments received by the fitDSTP function for fitting the DSTP model to human data

Argument	Description	Default
data	A data frame containing human data. See ?exampleData for data formatted correctly.	None
conditionName	If there is an additional experimental manipulation other than target congruency the model can only fit one at a time. Tell the function which condition is currently being fit. By default, the function assumes no additional condition (e.g., conditionName = NULL)	NULL
parms	A vector of starting parameters to use in the optimisation routine. Must be in the order: A, C, muT, muFL, muSS, muRS2, ter	c(0.15, 0.08, 0.10, 0.07, 0.325, 1.30, 0.24)
cdfs	A vector of quantile values for cumulative distribution functions to be estimated from the human data.	c(0.1, 0.3, 0.5, 0.7, 0.9)
cafs	A vector of quantiles for conditional accuracy functions to be estimated from the human data.	c(0.25, 0.50, 0.75)
maxParms	A vector containing upper limits on parameter values.	c(1, 1, 1, 1, 1, 2, 1)
nTrials	An integer stating how many trials to simulate per iteration of the fitting cycle for each congruency type.	50000
multipleSubjects	A boolean statement whether the fit is to multiple subjects (multipleSubjects = TRUE) or to a single subject (multipleSubjects = FALSE)	TRUE

Most of these arguments can be left at their default value, but changes will need to be made to the arguments `data`, `conditionName`, and `parms`. The time taken to fit the model to the data depends on several factors, but most notably how many trials are simulated per iteration of the optimisation routine. This is set to 50,000 trials by default (set by the `nTrials` argument). Generally, the more trials simulated the better, as this ensures the stochasticity inherent in each simulated run does not systematically affect the model's prediction. However, too many trials can make the fit routine rather slow.

In this example, we will fit the "present" condition from the "exampleData" data set, using the default starting model parameters. (Note that although some default starting parameters are provided, these should not be considered default in the sense that they are unchangable, fixed parameters, derived from multiple investigations and theory; they are just a starting point from which researchers should explore further.) We will assign the fitting procedure to the variable "fitPresent" by typing the following:

```
> fitPresent <- fitDSTP(data = exampleData, conditionName = "present")
```

Note that if the user wishes to retain the default function values, then these arguments do

not need to be provided. Once the optimisation routine has finished, the user can view the output by typing the variable name that the fit was stored to:

```
> fitPresent
```

The output provides three pieces of information: A vector of the best-fitting parameters, the value for the final minimised G^2 statistic, and the bBIC value for the current fit. To view the plot of the current model fit to the human CDF and CAF data, use the `plotFitDSTP` function (again, type `?plotFitDSTP` for detailed help on this function). This function requires the variable that the user stored the model fit to, the name of the data being fit, and the name of the condition being fit. (Note, if the user also changed other arguments from their default settings during the optimisation routine these will need to be passed to the plot function, too.) To plot the fit just conducted, type

```
> plotPresent <- plotFitDSTP(modelFit = fitPresent, data = exampleData,
conditionName = "present")
```

which displays the plot in R-Studio, and stores the data that made the plot into the variable `plotPresent`.

Avoiding Local Minima. When fitting models to data, it is important to ensure as best as possible that the optimisation routine has not ended in a *local minima*, which is an area of the parameter space where the objective function (i.e., G^2) is lower than in comparison to areas in the parameter space immediately surrounding its current location, but this area is not the lowest possible in the whole parameter space. The goal of parameter optimisation is to find this latter *global minima* whilst attempting to avoid local minima (for an excellent overview of the problem of parameter optimisation, see Lewandowsky & Farrell, 2010). Even if the current fit is quite good, one must ensure that it does not represent a local minima.

There is no consensus how best to deal with the potential for local minima, but one common method is to repeat the optimisation routine from multiple starting points; if the current fit represents a global minima, then the same final parameters should be reached from multiple starting points. This can be achieved in `flankr` by declaring a new variable with the new starting parameters in it, and then using this in the pass to `fitDSTP`:

```
> newParms <- c(0.08, 0.11, 0.13, 0.02, 0.365, 1.14, 0.28)
```

```
> fit <- fitDSTP(data = exampleData, conditionName = "present", parms =
newParms)
```

A broad—and automated—search of an area of the parameter space can be conducted using the `fitMultipleDSTP` function. The arguments to pass to this function are similar to those used by `fitDSTP`. It accepts a vector of starting parameters, as well as two new arguments: `var` and `nParms`. The function generates a set of random parameter values (with the number determined by the argument `nParms`). Each new vector of parameters is generated by sampling from a normal distribution with mean equal to the starting parameters, and standard deviation related to the `var` argument. Specifically, the standard deviation, SD, for each parameter is calculated as

$$\text{SD} = \left(\frac{StartingParameterValue}{100}\right) * var. \tag{6}$$

The function then explores each of these new parameter sets by fitting the model with them, and it stores the best fitting set as it progresses. The best fitting set of parameters after all have been explored is then returned to the user. Thus, if the user wishes to explore a range of parameters very close to the current set, they should set *var* to be very low; a broader parameter search from the starting parameter value will require a larger *var* setting.

```
> currentParms <- c(0.08, 0.11, 0.13, 0.02, 0.365, 1.14, 0.28)

> fit <- fitMultipleDSTP(data = exampleData, conditionName = "present",
parms = currentParms, var = 10, nParms = 20)
```

Note that this function can be quite slow, depending on how many new parameter sets to try (set by `nParms`) as well as how many trials to simulate per iteration of the fit routine (as before, this is set by the `nTrials` argument). If users wish to explore a wide range of parameter values (i.e., to get a coarse overview of the parameter space), then increase the value of `var`; to speed the process of searching a larger number of starting parameters, users may wish to reduce the number of trials (using the `nTrials` argument) and increase the value of the `nParms` argument:

```
> fit <- fitMultipleDSTP(data = exampleData, conditionName = "present",
parms = currentParms, var = 20, nParms = 100, nTrials = 2000)
```

If the user has reduced the number of trials per iteration in this fashion, it is advisable to run a final fit using the best-fitting parameters stored in the `fit` variable as input into the standard `fitDSTP` function with `nTrials` set to a larger value.

Fine-tuning the fitting procedure. As mentioned, there is no consensus in the literature in regards to best practice in finding best-fitting parameters. One method which I have found to work well often is to begin the fitting routine using single guesses of best-fitting parameter values using the `fitDSTP` function with relatively few trial numbers (set by the `nTrials` argument) to speed the process, and plotting each guess attempt (using the `plotFitDSTP` function) until parameters are found which produce predictions roughly corresponding visually to the human data. At this point, I use these parameters as a starting point in the `fitMultipleDSTP` function, with `nTrials` set relatively low (~5,000) to explore this general region of the parameter space. I then use the best-fitting parameters from this routine into a final call to `fitDSTP` . For example, if through trial-and-error with plotting I have a good set of starting parameters stored in the variable `bestParms`, I would use the following steps:

```
> newFit <- fitMultipleDSTP(data = exampleData, conditionName = "present",
parms = bestParms, var = 10, nParms = 40, nTrials = 5000)

> finalFit <- fitDSTP(data = exampleData, conditionName = "present", parms
= newFit$bestParameters, nTrials = 100000)
```

The SSP Model. The SSP model comprises 5 parameters (see Table 1) which need to be estimated from the human data. Functions that employ the SSP model are outlined in Table 4 and are very similar to those used by the DSTP model, with the exception of the name changes. The model simulation, fitting, and plotting procedures are also identical. It is my experience that fitting the SSP model is slower than fitting the DSTP model.

Table 4
Overview of the functions associated with the SSP model. To explore these functions in more detail, type ?functionName into the R console, where "functionName" is the name of the function detail is required for (e.g., ?fitSSP).

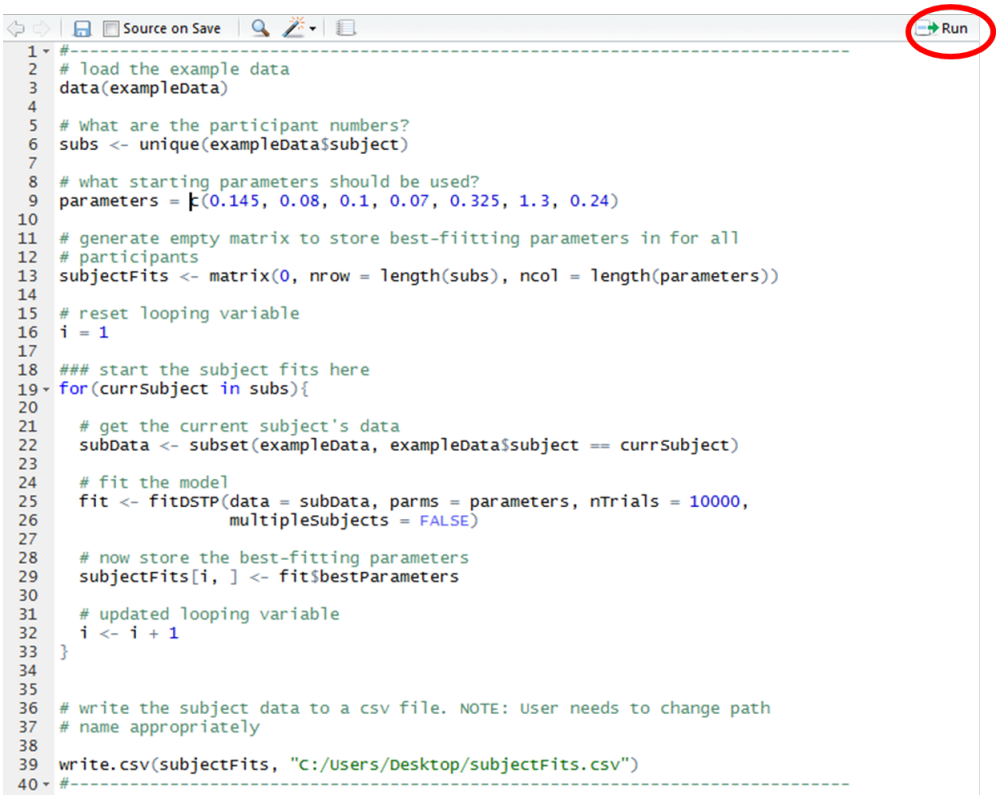
Function Name	Explanation
simulateSSP	Generate simulated data from the SSP model. Returns RTs (in seconds) and Errors for congruent and incongruent trials.
fitSSP	Fit the SSP model to human data. Returns the best-fitting parameters, as well as the G^2 statistic and the $bBIC$ statistic.
plotFitSSP	Plot the fit of the SSP model. This returns a plot of CDF and CAF for the data being fit, as well as that of the model with the best-fitting parameters found via the fitting routine. This function also returns distributional information from the human data and the model fit, so the user can use their own plotting software.
fitMultipleSSP	Fits the SSP model using multiple parameter starting points. The function returns the best-fitting parameters found via the search.

Examining Differences Between Conditions

What has not been discussed in this paper so far is how to test whether parameters in one model change as a function of an additional experimental manipulation. For example, the data set shipped with `flankr` not only manipulated the congruency of the flanker, but also manipulated whether there was a warning signal presented before stimulus onset or not (e.g., "present" vs. "absent"). Users might wish to examine whether presence of additional experimental manipulations significantly alters model parameters.

`flankr` does not provide any functions for this type of testing, so as to remain neutral as to which approach to use. It should be noted, though, that the functions that are built in to `flankr` can be used to accommodate most methods. For example, users may wish to fit one of the models to individual subject data (rather than group data, as demonstrated in the current paper) for each experimental condition, and then use standard inferential statistics on the parameter differences between conditions to check whether parameters are affected by experimental manipulations. Figure 7 shows a script users could alter to accommodate this method (making use of the `multipleSubjects = FALSE` argument being passed to the `fitDSTP` function). Remember, to explore multiple parameters per subject, users can consider the `fitMultipleDSTP` function. To run the script, users should press the `Run` button in the top-right of pane 1 (circled in Figure 7).

Another method is that of bootstrapping parameter estimates (Lewandowsky & Farrell, 2010). In this method, a model is initially fit to each additional condition using the methods described above to find the best-fitting parameters for each condition. These best-fitting parameters are then used to simulate new data with the same number of trials used in the experiment being fit (e.g., 250 congruent trials, 250 incongruent trials), and the model is fit to this simulated data. The best-fitting parameters from this fit are then stored, and the process is repeated N times, where N refers to how many bootstraps required. These boootstrapped parameter estimates can then be used to assess the variance in the parame-




```
#-------------------------------------------------------------------------------
# load the example data
data(exampleData)

# what are the participant numbers?
subs <- unique(exampleData$subject)

# what starting parameters should be used?
parameters = c(0.145, 0.08, 0.1, 0.07, 0.325, 1.3, 0.24)

# generate empty matrix to store best-fiitting parameters in for all
# participants
subjectFits <- matrix(0, nrow = length(subs), ncol = length(parameters))

# reset looping variable
i = 1

### start the subject fits here
for(currSubject in subs){

  # get the current subject's data
  subData <- subset(exampleData, exampleData$subject == currSubject)

  # fit the model
  fit <- fitDSTP(data = subData, parms = parameters, nTrials = 10000,
                 multipleSubjects = FALSE)

  # now store the best-fitting parameters
  subjectFits[i, ] <- fit$bestParameters

  # updated looping variable
  i <- i + 1
}


# write the subject data to a csv file. NOTE: User needs to change path
# name appropriately

write.csv(subjectFits, "C:/Users/Desktop/subjectFits.csv")
#-------------------------------------------------------------------------------
```

Figure 7. Sample script for fitting the DSTP model to individual subject data using the `fitDSTP` function.

ter estimates and for inferential testing . Figure 8 shows an example script bootstrapping parameters for one experimental condition.

Fixing Model Parameters

The fitting routines discussed in this paper consider all parameters to be free parameters—that is, during the fitting routine, each parameter of either model is free to vary its value when finding the best fit to data. Sometimes, though, users may wish to fix certain parameters, allowing only a subset of all parameters to be free to vary during the optimisation procedure.

This is implemented in `flankr` using two new functions: `fitDSTP_fixed` and `fitSSP_fixed` for fitting the DSTP and SSP models with fixed parameters, respectively. These functions are almost identical to the `fitDSTP` and `fitSSP` functions, but receives an additional variable `fixed`. `fixed` is a vector the same length as the number of parameters the model being fit (i.e., 7 for the DSTP model and 5 for the SSP model). Each element in this vector refers to each parameter in the model as specified in the argument `parms` that is sent to the fitting function (see Table 3). If the element is set to TRUE, then the parameter in that position will be fixed to the value set in the `parms` argument; if it is set to FALSE, the parameter in that position will be considered a free parameter.

For example, when fitting the DSTP model, if the user wishes to fix the A param-

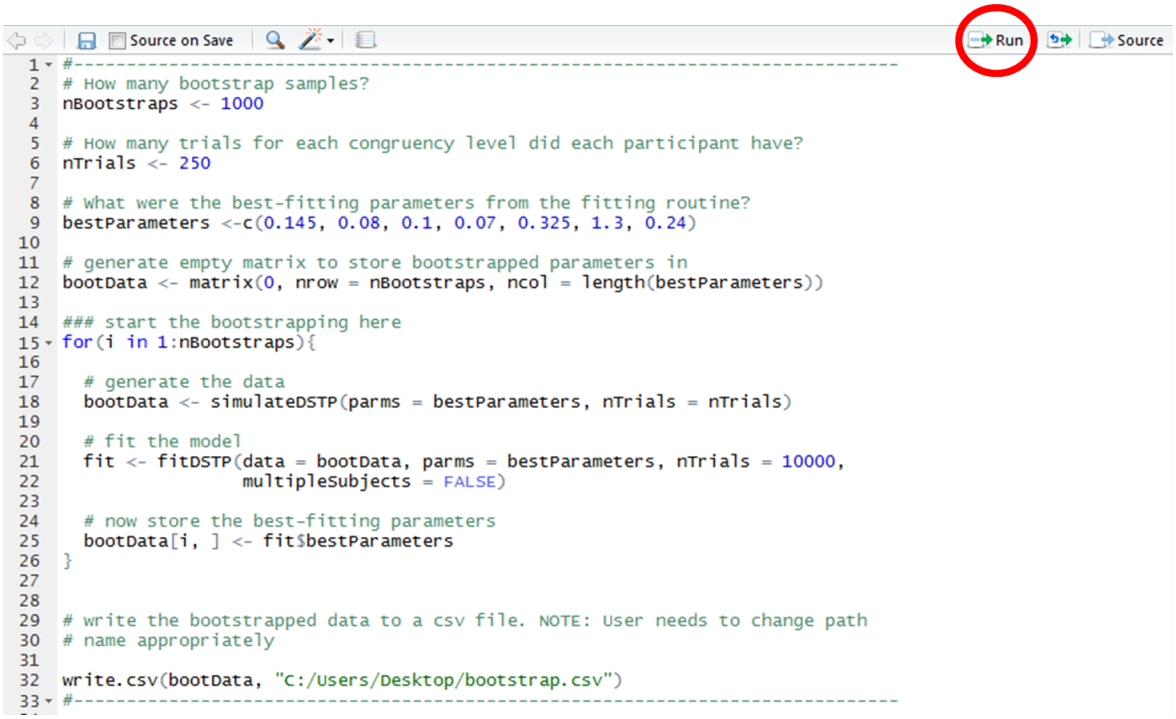


Figure 8. Code for conducting bootstrapping of parameter estimates. See text for details.

eter (which is in the first position of the `parms` argument), one would need to change the corresponding position in the `fixed` vector to TRUE:

```
> parms <- c(0.15, 0.08, 0.10, 0.07, 0.325, 1.30, 0.25)
> fixed <- c(TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE)
> fit <- fitDSTP_fixed(data = exampleData, conditionName = "present", parms
= parms, fixed = fixed)
```

During the fitting routine, any parameter indicated as TRUE in the `fixed` argument vector will be considered fixed and will not vary during optimisation. Note that the bBIC statistic will automatically reflect the reduced number of free parameters, as the statistic penalises models based on the number of free parameters. For more information about these functions, see the help files (`?fitDSTP_fixed` and `?fitSSP_fixed`). Note also that there are versions of these functions which still allow the user to explore multiple starting points of free parameters during optimisation (see `?fitMultipleDSTP_fixed` and `?fitMultipleSSP_fixed` for more information.)

Concluding Remarks

The DSTP and SSP models have advanced theorising regarding the nature of attentional selectivity in the flanker task by formalising how selectivity changes during stimulus processing in computional models. It is hoped that the package presented in the present paper will help researchers engage with these competing accounts by making the method of fitting the models to human data as simple as possible. The methods provided allow users

to simulate, fit, and / or plot the DSTP model and SSP model. Users can use the bBIC value returned by model fits to determine which model fits their data best.

References

Eddelbuettel, D., & François, R. (2011). Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, *40*(8), 1-18. Retrieved from `http://www.jstatsoft.org/v40/i08/`

Eriksen, B. A., & Eriksen, C. W. (1974). Effects of noise letters upon the identification of a target letter in a nonsearch task. *Perception & Psychophysics*, *16*, 143-149. doi: 10.3758/BF03203267

Eriksen, C. W., & St.James, J. D. (1986). Visual attention within and around the field of focal attention: A zoom lens model. *Perception & Psychophysics*, *40*, 225-240. doi: 10.3758/BF03211502

Farrell, S., & Lewandowsky, S. (2010). Computational models as aids to better reasoning in psychology. *Current Directions in Psychological Science*, *19*, 329-335. doi: 10.1177/0963721410386677

Fum, D., Missier, F. D., & Stocco, A. (2007). The cognitive modeling of human behavior: Why a model is (sometimes) better than 10,000 words. *Cognitive Systems Research*, *8*, 135-142. doi: 10.1016/j.cogsys.2007.07.001

Grange, J. A., & Houghton, G. (2014). Models of cognitive control in task switching. In J. A. Grange & G. Houghton (Eds.), *Task switching and cognitive control* (p. 160-199). New York, NY: Oxford University Press.

Gratton, G., Coles, M. G. H., Sirevaag, E. J., & Eriksen, C. W. (1998). Pre- and poststimulus activation of response channels: A psychophysiological analysis. *Journal of Experimental Psychology: Human Perception and Performance*.

Heitz, R. P., & Engle, R. W. (2007). Focusing the spotlight: Individual differences in visual attention control. *Journal of Experimental Psychology: General*, *136*, 217-240. doi: 10.1037/0096-3445.136.2.217

Hübner, R., Steinhauser, M., & Lehle, C. (2010). A dual-stage two-phase model of selective attention. *Psychological Review*, 759-784. doi: 10.1037/a0019471

Hübner, R., & Töbel, L. (2012). Does attentional selectivity in the flanker task improve discretely or gradually? *Frontiers in Psychology*, *3*, 434. doi: 10.3389/fpsyg.2012.00434

Jonides, J. (1983). Futher toward a model of the mind's eye movement. *Bulletin of the Psychonomic Society*, *21*, 247-250. doi: 10.3758/BF03334699

Lewandowsky, S., & Farrell, S. (2010). *Computational modeling in cognition.* Thousand Oaks, CA: Sage.

R Core Team. (2014). R: A language and environment for statistical computing [Computer software manual]. Vienna, Austria. Retrieved from `http://www.R-project.org/`

Ratcliff, R., & Smith, P. L. (2004). A comparison of sequential sampling models for two-choice reaction time. *Psychological Review*, *111*, 333-367. doi: 10.1037/0033-295X.111.2.333

White, C. N., Ratcliff, R., & Starns, J. J. (2011). Diffusion models of the flanker task: Discrete versus gradual attentional selection. *Cognitive Psychology*, 210-238. doi: 10.1016/j.cogpsych.2011.08.001